Vysoká škola báňská – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Absolvování individuální odborné praxe
Individual Professional Practice in the Company

2014 **Miroslav Lazar**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Miroslav Lazar**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: KVADOS, a.s.
2. Struktura závěrečné zprávy:
a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta
b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti
c) Zvolený postup řešení zadaných úkolů
d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe
e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe
f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vedl odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Pavel Moravec, Ph.D.**

Konzultant bakalářské práce: Ing. Radek Garzina, Ph.D.

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: *2. 5. 2014*

……..……

podpis studenta

Poděkování

Rád bych poděkoval Ing. Radkovi Garzinovi, Ph.D za možnost absolvování odborné praxe, projevenou důvěru, přívětivý přístup a trpělivost. Ing. Pavlu Moravcovi, Ph.D. za odborné vedení, flexibilitu a připomínky. Kolektivu firmy KVADOS za přátelskou atmosféru, Markétě za inspiraci a přátelům a rodině za podporu, která přispěla ke zdárnému dokončení práce.

Prohlášení zástupce spolupracující právnické nebo fyzické osoby

„Souhlasím se zveřejněním této bakalářské/diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských/magisterských programech VŠB-TU Ostrava.“

Dne: *2. 5. 2014*

……
podpis zástupce

Abstrakt

Tato práce popisuje mojí činnost ve firmě KVADOS a.s. zabývající se činností na poli vývoje softwaru. Náplní práce bylo vyvinout nástroj pro migraci dat v podobě úkolů pro jednotlivé zaměstnance z ERP systému VENTUS do Team Foundation Server na platformě .NET s využitím programovacího jazyka C#, technologie ASP.NET a jeho rozšíření pomocí knihovny Ext.NET.

Klíčová slova

.NET; C#; ASP.NET; Ext.NET; JavaScript; Team Foundation Server; Ventus;

Abstract

This document describes my work for the company KVADOS a.s. engaged in activities in the field of software development. The aim of work was to develop a tool for migrating data from ERP system VENTUS to Team Foundation Server on .NET platform using the C# programming language, ASP.NET and his extension called Ext.NET library.

Key words

.NET; C#; ASP.NET; Ext.NET JavaScript; Team Foundation Server; Ventus;

Seznam použitých zkratek

Zkratka	Význam
AD	Active Directory
TFS	Team Foundation Server
VS	Visual Studio
JS	JavaScript
C#	C Sharp

Obsah

1. Úvod

Tématem této práce je seznámit čtenáře s náplní mé práce ve společnosti KVADOS a.s. v rámci odborné praxe během bakalářského studia. Cílem po celou dobu praxe bylo vytvořit zcela nový nástroj pro usnadnění rozdělování práce jednotlivým zaměstnancům z výrobního oddělení. Na tomto projektu jsem spolupracoval se spolužákem Karlem Antošíkem. Zadaný úkol byl po dobu vývoje rozdělen na dva nezávislé celky. Každý z nás se proto mohl bez omezení věnovat náplni své práce. Pro celistvý pohled na věc bych chtěl tímto na jeho práci odkázat. Požadavkem bylo vytvořit základní verzi nástroje pro usnadnění některých interních postupů na výrobním oddělení firmy, který bude i po ukončení naší praxe následně dále rozšiřován a nasazen do provozu.

V úvodu naleznete stručné seznámení se společností KVADOS a. s. a jejího vlastního produktu VENTUS. Druhá a třetí kapitola obsahuje zadání a seznam jednotlivých fází práce řešených po celou dobu trvání odborné praxe. Čtvrtá kapitola podrobně popisuje způsob řešení jednotlivých podúkolů a problémy, které se během tvorby nečekaně vyskytly. Na závěr celé práce přikládám své hodnocení nabytých znalostí ze studia vysoké školy a jejich využití na této odborné praxi a celkové zhodnocení vykonané práce.

1.1 O společnosti Kvados a.s.

KVADOS a.s. je středoevropským producentem a dodavatelem vlastních softwarových řešení. Vyskytuje se na trhu od roku 1992 a zaměřuje se hlavně na klienty ze segmentu obchodu a služeb. Těm poskytuje vlastní softwarová řešení pod značkami *VENTUS®, myAVIS™, myFABER™, myWORK™, mySTOCK™, myTEAM™, myMACHINE™, myCASH™ a myDATACENTER™* Společnost se samá dále prezentuje jako:

„KVADOS a. s. si zakládá na vysoké kvalitě vnitřních procesů, které má certifikovány dle všech 6 norem využitelných v oblasti informačních a komunikačních technologií.

Dlouhodobě vyhledává příležitosti k vytváření nových trhů, kde si následně buduje pozici lídra. Podařilo se jí to například u mobilních informačních systémů. Stala se respektovaným hráčem již v 11 evropských zemích (obrázek 1.1) a svými produkty mnohde přispěla nejen k lepším hospodářským výsledkům svých zákazníků, ale i k optimalizaci procesů a změn myšlení lidí.

KVADOS, a.s., zůstává českou nezávislou akciovou společností s centrálou v Ostravě. Snaží se být dobrým sousedem, zapojuje se do lokálního dění a podporuje kvalitní projekty. Je také zakládajícím členem IT Clusteru – sdružení, které je vůdčí silou výzkumu a vývoje nových technologií v Moravskoslezském kraji.“ [1]

Obrázek 1.1: Zastoupení společnosti KVADOS a. s. ve světě

1.2 Popis pracovního zařazení studenta

Společnost KVADOS a.s. sídlí v centrále nacházející se v Ostravě – Mariánských Horách. Tato odborná praxe byla absolvována na jednom z výrobních oddělení na pozici stážista. Přesněji na .NET výrobním oddělení zodpovědné za velkou část produkce společnosti KVADOS a.s. pod vedením .NET Developer Team Leadera Ing. Radka Garziny Ph.D.

2. Řešená problematika

Většina činnosti spočívala v okamžité implementaci kódu pro realizaci požadovaných funkcí pomocí agilních metodik, kdy byl vznesen požadavek na určitou funkci a ta byla po krátké analýze ihned implementována a odzkoušena. Po schválení funkčnosti se přistoupilo k rozšíření aplikace o další funkci nebo byla v případě připomínek stávající upravena, opět odzkoušena, schválena a mohlo se opět přistoupit k dalšímu úkolu.

2.1 Ventus

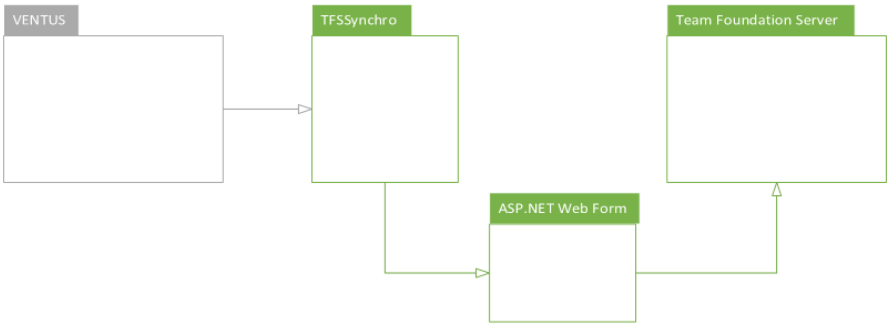
Pro pochopení řešené problematiky a ujasnění si postupů práce je třeba se seznámit s ERP systémem VENTUS. Společnost KVADOS a.s toto své řešení pro zákazníky popisuje následovně:

„Jedním z produktů společnosti KVADOS a.s. je i informační systém VENTUS®, který je otevřeným modulárním programovým systémem ERP pro realizaci komplexního řešení v obchodních, obchodně-výrobních, distribučních a logistických společnostech. VENTUS® je schopen pokrýt všechny procesy probíhající ve společnostech tohoto segmentu včetně speciálních modulů pro celní agendy, materiálové řízení výroby, řízení skladového hospodářství, plánování, MIS a CRM nebo napojení na rozhraní B2B i B2C. Informační systém VENTUS® je určen středním a větším společnostem, které požadují ověřený celopodnikový informační systém (ERP) přinášející prokazatelné zvýšení efektivity a transparentnosti činností a procesů, udržení manažerské kontroly při realizaci velkého objemu činností, se specializací na oblast logistiky a distribuce zboží. Je vhodný zejména pro společnosti, jejichž distribuční a obchodní model vyžaduje nestandardní řešení. Zákazníci informačního systému VENTUS® nemusejí přizpůsobovat své procesy ERP systému, ale jsou schopni informační systém přizpůsobit vlastním procesům.“ [2]

2.2 Zadaní náplně práce

Tento systém si společnost také upravuje pro vlastní obchodně-výrobní činnost napříč všemi odděleními při řízení personalistiky u dlouhodobých projektů. Díky tomuto systému může být řešený projekt rozdělen na hlavní úkoly pro jednotlivé výrobní týmy zodpovědné za určitou část řešení. V tomto týmu jsou poté rozděleny jednotlivé dílčí úkoly konkrétním pracovníkům s požadovaným termínem vypracování. Programátoři při své práci však využívají řešení Microsoft Team Foundation Server, umožňující mimo jiné i řízení cyklu vývoje softwaru a management zdrojových kódů. Díky paralelnímu využití těchto dvou systémů dochází k určité desynchronizaci a práci navíc, protože vedoucí pracovníci musí veškeré změny mezi systémy manuálně přenášet.

Tento způsob práce není při dlouhodobém vývoji softwarových řešení nejšťastnější. Při zadávání nových nebo změnách stávajících úkolů je potřeba tato data ze systému VENTUS přenášet do TFS. Po splnění požadovaných úkolů programátory je vyžadováno opět tyto změny zaznamenat do systému VENTUS, kde k nim přistupují ostatní členové týmu podílející se na vývoji. Dále je třeba z důvodu personalistiky evidovat, kdy a kdo byl v rámci celkového vývoje zodpovědným řešitelem jednotlivých úkolů, a z těchto záznamů vytvářet reporty.



Obrázek 2.1: Náčrt řešení s zvýrazněnou vyvíjenou částí

Cílem praxe nebylo vytvořit kompletní verzi tohoto nástroje, který by vše řešil. Našim cílem bylo zprovoznění základní funkce migrace zadaných úkolů z VENTUS do TFS (Obrázek 2.1). Spolupracovník Karel Antošík řešil část, která obstarávala získávání úkolů z interního systému VENTUS. Cílem mého působení bylo vyvinout nástroj pro správu takto získaných úkolů a jejich následnou migraci do TFS. Této problematice se budu na následujících stranách věnovat.

3. Seznam zadaných úkolů a jejich časová náročnost

Úkol k vypracování	Časová náročnost
Seznámení se s firemním využitím TFS a VENTUS	4 dny
Příprava uživatelského typu úkolu	3 dny
Komunikace s TFS pomocí C#	3 dny
Vkládání a získávání položek z TFS	8 dnů
Tvorba rozhraní pro komunikaci s TFS	4 dny
Příprava společné relační databáze	2 dny
Tvorba rozhraní pro komunikaci s relační databází	2 dny
Příprava funkce synchronizace	2 dny
Seznámení se s Ext.NET	4 dny
Tvorba základního GUI a zobrazení dostupných úkolů	2 dny
Stránkování dostupných úkolů	1 den
Přesun úkolů s možností volby projektu	2 dny
Detailní informace o úkolu	2 dny
Skrytí zvoleného úkolu a opětovné zobrazení	2 dny
Filtrace zobrazených záznamů dle zadaných kritérií	4 dny
Obohacení filtru o funkci „zobraz vše“	2 dny
Možnosti ověření identity v ASP.NET s využitím AD	1 den
Tvorba přihlašovacího GUI	1 den
Modifikace tříd pro ověření přístupu k projektu	2 dny
Modifikace filtru pro spolupráci s ověřením přístupu	3 dny
Sloučení vyvíjených projektů do společného řešení	2 dny
Plán nasazení	1 den

Tabulka 1: Seznam úkolů

4. Zvolený postup řešení zadaných úkolů

Na následujících stranách této kapitoly je detailně popsán postup řešení jednotlivých zadaných úkolů. Dále se věnuje problémům, jenž se během prací vyskytly a vypořádání se s nimi. Kapitola popisuje úkoly, které lze rozdělit do čtyř okruhů technologií a jejich vzájemné provázání do jednoho funkčního celku. Při tvorbě nástroje se využilo technologie .NET a jeho assemblies pro komunikaci s TFS, relační databáze SQL SERVER pro uložení přenášených dat, ASP.NET a jeho rozšíření v podobě knihovny Ext.NET pro tvorbu uživatelského rozhraní a Active Directory pro autentizaci.

4.1 Team Foundation Server

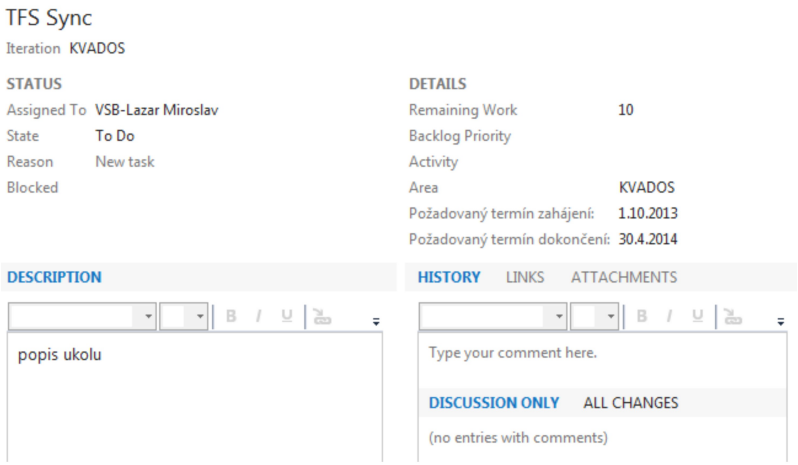
Během vývoje bylo potřeba propojit několik technologií. Většina byla z dílen Microsoftu založených na technologií z rodiny .NET v čele s programovacím jazykem C#. Vývoj probíhal v té době na poslední verzi vývojového prostředí Visual Studio 2013 Professional.

Prvním úkolem bylo seznámit se s nejnovější verzí produktu Team Foundation Server 2013 [3] pro jeho plánované nasazení a s možnostmi strojového získávání a ukládání dat. Díky zasvěcení do využití TFS v rámci výrobních postupů firmy vedoucím Ing. Radkem Garzinou Phd. jsem si vytvořil základní představu o řešené problematice a začal studovat možnosti TFS.

TFS umožňuje správu členů vývojářského týmu, centrální uložení vyvíjeného kódu a rozdělování úkolů v rámci určitého projektu podle dnes nejpopulárnějších šablon způsobu práce v týmu. TFS je zakomponován do Visual Studia a přes něj nebo webové rozhraní umožňuje přidávání úkolů jednotlivým členům vývojového týmu.

Mezi nejznámější modely kvality organizace práce patří metody CMMI a Scrum. TFS tyto metodiky plně podporuje. CMMI je určená spíše velkým firmám s přesně danými postupy a cíli, kterých musí být napříč všemi procesními oblastmi dosaženo. Od analýzy až po implementaci. Scrum je naopak agilní metodika vhodná pro menší týmy s volnějším způsobem práce. Vyvíjené řešení je rozděleno na jednotlivé funkce a zařazeno podle důležitosti do tzv. Iterací. Kdy po dokončení každé iterace je vždy k dispozici funkční část celého řešení. V případě nevyhovění funkčních požadavků je funkce v další iteraci přepracována a opět zkontrolována. Tohoto způsobu práce bylo také využito při tvorbě dále popisovaného nástroje.

Přes VS nebo webové rozhraní uživatel standardně zvolí typ úkolu, zadá instrukce, co je potřeba a do kdy dodělat, a nakonec určí programátora, který tento úkol dostane na starosti. Programátorovi je po otevření projektu z centrálního repositáře tento úkol zobrazen (Obrázek 4.1) a po splnění nejlépe v zadaném termínu ho programátor uzavře a vykáže tak svojí práci.



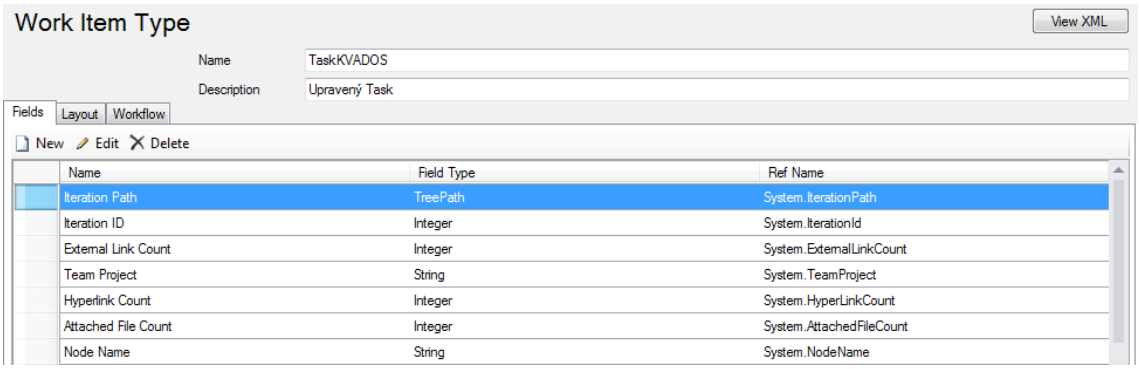
Obrázek 4.1: detail úkolu ve VS

4.2 Příprava uživatelského typu úkolu

TFS obsahuje několik předem definovaných typů úkolů jako jsou Task, Bug a další. Avšak každá firma může vyžadovat jiný formát obsažených informací v jednotlivých typech úkolů. Proto TFS umožňuje (nejlépe přes VS) tvorbu dalších uživatelsky definovaných typů úkolu.

Microsoft nabízí tři verze TFS. Team Foundation Services (nově Visual Studio Online) formou cloudového řešení s pohodlným přístupem odkudkoliv, Team Foundation Server Express zdarma pro instalaci na lokální počítače a komerční plnohodnotnou verzi Team Foundation Server. Při zjišťování možností TFS bylo zjištěno jednoho zásadního rozdílu mezi verzemi TFS. U cloudové verze, která je jinak velice uživatelsky přívětivá, se ukázalo, že neumožňuje vytvoření nového uživatelsky definovaného typu úkolu. Pro další práci naprosto zásadní věc. Po vyřešení tohoto problému přechodem na verzi Express stačilo vybrat způsob vytvoření nového typu úkolu.

Visual Studio nabízí pro import nového typu úkolu ve formátu XML konzolovou utilitu witadmin. Pro lepší efektivitu práce byl nakonec použit doplněk pro VS jménem Visual Studio Team Foundation Server 2013 Power Tools přidávající možnost pohodlného exportu a importu uživatelských typů úkolu včetně jejich editace pomocí GUI (Obrázek 4.2).



Obrázek 4.2: Editace WorkItem pomocí TFS 2013 Power Tools ve VS

4.3 Komunikace s TFS pomocí C#

Pro přístup k třídám pro komunikaci s TFS pomocí programovacího jazyka C# musíme přidat reference na následující assemblies [5]:

```
Microsoft.TeamFoundation.Client;
Microsoft.TeamFoundation.Common;
```

Poté pro přístup ke kolekci úkolů uložených v TFS využijeme tento kód:

```
Uri collectionUri = new Uri(uriTfsString);
TfsTeamProjectCollection teamProjectCollection = new
TfsTeamProjectCollection(collectionUri);
WorkItemStore workItemStore = teamProjectCollection.GetService
<WorkItemStore>();
```

4.4 Vkládání a získávání položek z TFS

4.4.1 Čtení dat z Team Foundation Server

Získávání dat z této kolekce lze řešit dvěma způsoby. Pokud víme přesně, který úkol chceme z TFS získat, předáme jeho identifikátor metodě `GetWorkItem` dotazované kolekce. Metoda vrátí námi požadovaný úkol typu `WorkItem`.

Z tohoto objektu typu `WorkItem` můžeme přes jeho vlastnosti získat jednotlivé údaje, které následně uložíme do námi navržené instance objektu představující náš specifický typ úkolu. Pro získání uživatelem nadefinovaných položek slouží vlastnost `Field`, které se jako index předá název reference daný při tvorbě položek vlastního uživatelského typu úkolu.

Druhým způsobem dotazování vůči TFS je pomocí syntaxe podobné jazyku SQL, přičemž si stačí pohlídat, aby dotaz splňoval tyto podmínky:

```
část dotazu ve tvaru: FROM WorkItems
část dotazu ve tvaru: WHERE [System.WorkItemType] = 'VÁŠ TYP'
```

Poté již stačí zavolat na `WorkItemStore` úkolů metodu `Query` a jako parametr jí předat váš SQL dotaz. Metoda vrací `WorkItemCollection` s požadovanými úkoly, kterou lze jednoduše projít pomocí cyklu a dále zpracovat. Způsob dotazování se pomocí syntaxe podobné SQL je vhodnější, když chceme získat úkoly splňující určitá specifika.

4.4.2 Zápis dat do Team Foundation Server

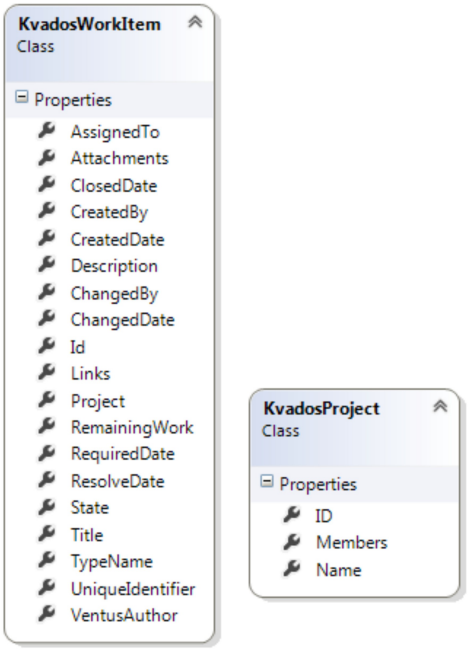
Při vytváření nového úkolu pro TFS musíme určit projekt, ke kterému nový úkol chceme přiřadit. Ten získáme přes instanci `WorkItemStore` a index její vlastnosti `Project`. Z něj pak získáme přes index vlastnosti `WorkItemType` typ úkolu, jenž má projekt k dispozici. Poté vytvoříme nový objekt typu `WorkItem` a v konstruktoru mu předáme typ získaného úkolu. Touto provázaností si nový úkol už sám odvodí, k jakému projektu patří. Do objektu typu `WorkItem` přes vlastnosti a speciální vlastnost `Fields` zadáme údaje o úkolu, jenž chceme vytvořit a na závěr úkol natrvalo uložíme do TFS zavoláním jeho metody `Save`. Při nastavování hodnot přes vlastnost `Fields` si musíme hlídat, zda námi tvořený úkol zvoleného typu skutečně obsahuje pole zadávaná přes její index. Jinak by se vyskytla chyba, že úkol není připraven k uložení.

Editace úkolu je kombinací všech předchozích postupů. Napřed si požadované úkoly vyhledáme pomocí dotazu SQL nebo přes identifikátor. Nastavíme hodnoty nové a takto upravený úkol jednoduše opět uložíme zavoláním metody `Save`.

4.5 Tvorba rozhraní pro komunikaci s TFS

Po nastudování všech potřebných postupů již nic nebránilo tvorbě základní kostry aplikace v podobě entitních tříd sloužících pro uchovávání přenášených údajů a tříd pro přístup k TFS.

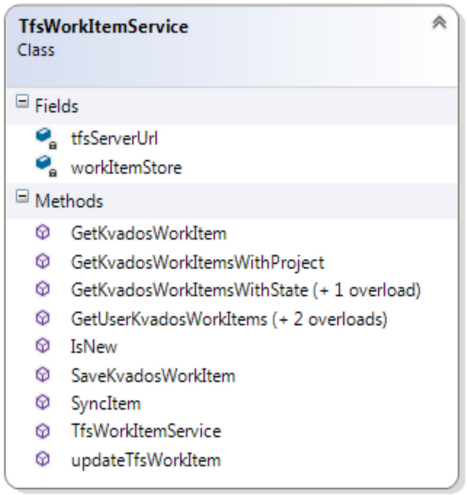
Hlavní entitní třída `KvadosWorkItem` (Obrázek 4.3) reprezentuje přenášený úkol, jenž obsahuje všechny vyžadované vlastnosti úkolu mezi nimiž je i položka zastupující další vytvořenou entitní třídu `KvadosProject` v podobě projektu, kam úkol zařadit. Některé vlastnosti v entitní třídě `KvadosWorkItem` nebudou zatím využity. Avšak plánuje se jejich použití při budoucím rozšiřování nástroje.



Obrázek 4.3: Entitní třídy a jejich vlastnosti

4.5.1 Metody pro načítání úkolů

Pro přístup k datům byla vytvořena třída pro přístup k jednotlivým úkolům zvoleného projektu. Ve třídě `TfsWorkItemService` (Obrázek 4.4) určené pro správu úkolů byly vytvořeny metody pro získávání kolekcí úkolů podle mnoha různých požadovaných parametrů (Obrázek 4.5). Zde se využilo dotazovacího jazyka se syntaxí podobnou SQL.



Obrázek 4.4: Třída pro práci s TFS

Na základě změn požadavků v průběhu vývoje byly některé tyto metody dodatečně modifikovány. Dále byly doplněny pro další funkčnost metody pro získání úkolu podle

zadaného identifikátoru TFS a metoda pro zjištění, zda je vybraný úkol aktuálnější, než jeho verze uložená v TFS.

```
public List<KvadosWorkItem> GetKvadosWorkItemsWithProject(string project)
{
    Dictionary<string, string> parameters = new Dictionary<string, string>();
    parameters.Add("project", project);

    Uri collectionUri = new Uri("TFS URL");
    TfsTeamProjectCollection teamProjectCollection = new TfsTeamProjectCollection(collectionUri);
    WorkItemStore workItemStore = teamProjectCollection.GetService<WorkItemStore>();

    Query queryWithState = new Query(workItemStore, @"SELECT [System.Id]
                                FROM WorkItems
                                WHERE [System.WorkItemType] = 'TaskKVADOS'
                                AND [System.TeamProject] = @project
                                ORDER BY [System.Id]", parameters);

    WorkItemCollection works = workItemStore.Query(queryWithState.QueryString);
    List<KvadosWorkItem> workItemsWithState = new List<KvadosWorkItem>();

    foreach (Microsoft.TeamFoundation.WorkItemTracking.Client.WorkItem item in works)
    {
        KvadosWorkItem kvadosWork = KvadosWorkItemFactory.CreateKvadosItem(item.Id, item.Title, item.Description,
        Convert.ToString(item.Type), Convert.ToString(item.Fields["Assigned To"].Value), item.State,
        Convert.ToDateTime(item.Fields["Termin Zahajeni"].Value), Convert.ToDateTime(item.Fields["Termin Dokonceni"].Value),
        Convert.ToDateTime(item.Fields["Created Date"].Value), Convert.ToDateTime(item.Fields["Changed Date"].Value),
        Convert.ToDateTime(item.Fields["Closed Date"].Value), Convert.ToString(item.Fields["Created By"].Value),
        Convert.ToString(item.Fields["Changed By"].Value), Convert.ToInt32(item.Fields["Remaining Work"].Value));
        workItemsWithState.Add(kvadosWork);
    }

    return workItemsWithState;
}
```

Obrázek 4.5: Metoda pro získání úkolů podle zadaného názvu projektu

4.5.2 Metody pro ukládání úkolů

Druhou část třídy tvoří metody pro uložení úkolů do TFS. Metoda `SaveKvadosWorkItem(KvadosWorkItem insertedWorkItem)` (Obrázek 4.6) uloží nový úkol do TFS projektu. Cílový projekt získá podle názvu projektu uloženém v úkolu. Vyhledá tento projekt ve `WorkItemStore` a k němu nový úkol přiřadí. Výchozí položky se nastaví přes klasické vlastnosti. Položky specifické pro konkrétní typ úkolu opět nastavíme přes vlastnost `Field` a její index. TFS identifikátor nově přidaného úkolu je metodou vrácen pro pozdější využití.

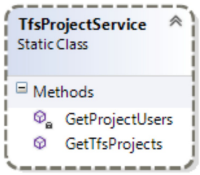
Další metoda `updateTfsWorkItem(KvadosWorkItem update)` slouží pro aktualizaci již uložených úkolů. Metoda přijímá dříve načtený úkol určený jeho identifikátorem, vyhledá tento úkol v TFS, změní obsah a opět tento úkol opět uloží.

Kompletní operaci uložení nového úkolu získaného z jiného úložiště sdružuje metoda `SynItem(KvadosWorkItem)`. Metoda byla připravena až po zavedení relační databáze jako zdroje ukládaných úkolů. Uloží úkol pomocí `SaveKvadosWorkItem` do TFS a díky vrácenému TFS identifikátoru ho sváže s úkolem v relační databázi přes třídu k tomu určenou.

```
public int SaveKvadosWorkItem(KvadosWorkItem insertedWorkItem)
{
    Project teamProject = workItemStore.Projects[insertedWorkItem.Project.Name];
    WorkItemType workItemType = teamProject.WorkItemTypes[insertedWorkItem.TypeName];

    WorkItem newWorkItem = new WorkItem(workItemType)
    {
        Title = insertedWorkItem.Title,
        Description = insertedWorkItem.Description,
        State = insertedWorkItem.State,
    };

    newWorkItem.Fields["Assigned To"].Value = insertedWorkItem.AssignedTo;
    newWorkItem.Fields["Termin Zahajeni"].Value = insertedWorkItem.RequiredDate;
    newWorkItem.Fields["Termin Dokonceni"].Value = insertedWorkItem.ResolveDate;
    newWorkItem.Fields["Remaining Work"].Value = insertedWorkItem.RemainingWork;
    newWorkItem.Fields["Zadavatel"].Value = insertedWorkItem.VentusAuthor;

    newWorkItem.Save();

    return newWorkItem.Id;
}
```

Obrázek 4.6: Uložení úkolu do TFS

Pro přístup k seznamu projektů uložených v TFS bylo potřeba vytvořit statickou třídu `TfsProjectService` (Obrázek 4.7). V základní verzi obsahovala metodu `GetTfsProjects` vracející `List` všech projektů v TFS. Tato třída byla později rozšířena pro novou požadovanou funkci.



Obrázek 4.7: Třída pro zisk projektů z TFS

4.6 Příprava společné relační databáze

Aby se předešlo nechtěným změnám v systému VENTUS, nepřistupovalo se přímo k datům uloženým v jeho databázi. Pro předávání dat ze systému VENTUS do TFS byla vytvořena relační databáze založena na Microsoft SQL Server 2012 sloužící jako logické oddělení obou systémů.

Výše zmíněný kolega Karel Antošík měl za úkol zpracovat data ze systému VENTUS a ukládat je do této databáze v předem dohodnutém formátu. Mým úkolem bylo tato data zpracovat a uložit do TFS.

Po přípravě všech potřebných nástrojů v čele s Microsoft Management Studio se mohlo přistoupit přímo k tvorbě další části vyvíjené aplikace. K dispozici byl již dříve připravený základní relační model. Model obsahoval hlavní tabulky, jenž jsou potřeba k přenosu námi požadovaných dat. Tento základní model jsme každý využívali jako lokální instanci SQL Serveru. Později byl tento model rozšířen o další atributy tabulek potřebné při rozšiřování funkčnosti celého nástroje. Využitý relační model obsahuje relace popsané v (Tabulka 2).

Tabulka	Popis
TfsProject	Pro plánované využití při rozšiřování v budoucnu, nebylo využito.
User	Číselník identifikující vazbu mezi uživatelskými jmény v obou systémech.
Workitem	Obsahuje úkoly určené pro přesun mezi systémy.
WorkitemHistory	Repositář všech změn úkolu, plánuje se v využití v budoucnu, nebylo využito.
WorkitemRels	Identifikuje jednotlivé úkoly v rámci VENTUS a jeho verzi v TFS.
WorkitemState	Číselník obsahuje výčet všech stavů jakých může úkol nabýt.
WorkitemType	Číselník obsahuje výčet všech typů úkolu.
Preferences	Obsahuje datum poslední synchronizace používané při identifikaci úkolů změněných po tomto datu.

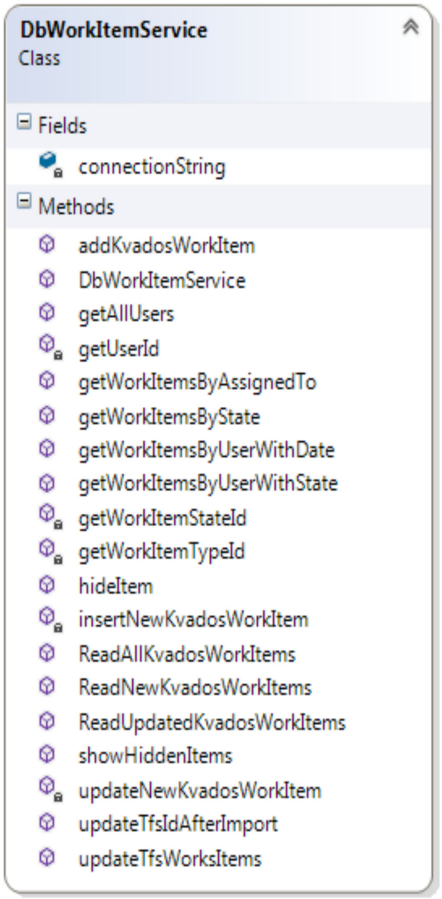
Tabulka 2: Seznam využitých relací

4.7 Tvorba rozhraní pro komunikaci s relační databází

Pro komunikaci s relační databází byla v rámci vyvíjené aplikace vytvořena třída `DbWorkItemService` (Obrázek 4.8). Tato třída obsahuje veškeré metody starající se o zápis a čtení z relační databáze SQL Server 2012 pomocí technologie ADO.NET a dotazovacího jazyka SQL.

Pro splnění hlavní funkcionality byla implementována celá řádka metod pro získání kolekcí úkolů načtených z relační databáze splňujících určité parametry. Všechny tyto metody funguji obdobně.

Vytvoří se díky instanci `SqlConnection` spojení s relační databází podle jejího `connectionString`. Následně je této instanci předán SQL dotaz předem uložený do objektu třídy `SqlCommand`. Této instanci je díky vlastnosti `Parameters` a její metody `Add` přidán nový parametr ve formě řetězce zastupující parametr SQL dotazu se shodným názvem začínající symbolem „@“ a stejným datovým typem. Vlastností `Value` metody `Add` je tomuto parametru přidělena jeho hodnota. Ve většině metodách pochází tyto hodnoty ze vstupních parametrů konkrétní metody.



Obrázek 4.8: Třída pro práci s relační databází

Výsledek SQL dotazu se zvolenými parametry je získán zavoláním metody `ExecuteReader` na instanci `SqlCommand`. Třída vrací objekt typu `SqlDataReader` obsahující námi požadovaná data. Jednotlivé vrácené řádky tabulky obsažené v instanci typu `SqlDataReader` se posléze projdou v cyklu díky metodě `Read` a získají jednou z mnoha metod se zadaným indexem sloupce jako jsou `GetString`, `GetInt32` a další. Zde je třeba hlídat, aby hodnota získávána pomocí indexu sloupce v určitém řádku byla shodného datového typu s kterou je metoda pro získání hodnoty kompatibilní. Takto získanými hodnotami se už jen naplní nová instance úkolu a uloží do kolekce typu `List`. Tu metoda nakonec celou vrací. Tyto metody slouží jako základ pro později vytvořenou webovou aplikaci pomocí technologie ASP.NET a jeho Ext.NET knihovny třetí strany.

Zvláštní zastoupení pro pozdější využití má metoda `ReadNewKvadosWorkItems` vracející list úkolů, které nemají nastavený atribut `tfs_id`, takže ještě nebyly do TFS nahrány. `List` těchto vrácených úkolů bude využit pro zobrazení úkolů, jenž jsou připraveny na migraci do TFS.

V některých metodách je potřeba podle zadaného názvu získat identifikátor, pod kterým je takto pojmenovaná položka v databázi uložena. K lepší přehlednosti kódu jsou metody pro tuto funkci připraveny odděleně. Metody vrací identifikátor zadaného uživatele, typu úkolu a stavu úkolu.

Důležitou funkci má metoda `UpdateTfsIdAfterImport(int tfsId, Guid id)`. Metoda se volá po úspěšném uložení úkolu do TFS, který se doposud nacházel jen v relační databázi. Po vložení úkolu do TFS vyhledá tento úkol v relační databází a nastaví mu `tfs_id` přidělené od TFS pro přesnou identifikaci mezi systémy a pro pozdější synchronizaci záznamů.

```
public List<KvadosWorkItem> ReadNewKvadosWorkItems()
{
    DatabaseService db = new DatabaseService(connectionString);
    db.Connect();

    SqlCommand cmd = db.CreateCommand("ReadNewKvadosWorkItems");
    cmd.CommandType = CommandType.StoredProcedure;

    SqlDataReader reader = db.Select(cmd);
    List<KvadosWorkItem> works = new List<KvadosWorkItem>();

    KvadosWorkItemFactory factory = new KvadosWorkItemFactory();

    while (reader.Read())
    {
        KvadosWorkItem newWork = new KvadosWorkItem();

        newWork = factory.createNewItem(reader);

        works.Add(newWork);
    }

    reader.Close();
    db.Close();

    return works;
}
```

Obrázek 4.9: Upravená metoda pro volání uložených procedur

Třídu `DbWorkItemService` po dokončení všech prací následně upravil spolupracovník Karel Antošík pro znalosti ze své části projektu. Veškeré SQL dotazy přesunul do uložených procedur na stranu databázového serveru a třídu přizpůsobil pro jejich volání (Obrázek 4.9). Tímto se dotazy staly nezávislými na kódu vyvíjeného nástroje v případě dodatečných úprav. Došlo také k centralizaci těchto dotazů na jedno místo, takže volané dotazy a jejich výsledky budou vždy pro všechny stejné.

4.8 Příprava funkce synchronizace

Po vytvoření všech stěžejních metod pro spolupráci s relační databází byla ještě pro pozdější možné využiti připravena funkce pro automatickou synchronizaci úkolů směrem do TFS . Metoda zodpovědná za tuto funkcionalitu poběží jako služba na pozadí a v intervalech bude kontrolovat změny v relační databázi a tyto pozměněné úkoly ihned ukládat do TFS pomocí třídy `TfsWorkItemService` a metody `updateTfsWorkItem`. Pro tuto

funkcionalitu byla využita v relační databázi tabulka `Preferences`, která uchovává záznam o času poslední aktualizace a po úspěšném provedení synchronizace je tento údaj opět přepsán. Z důvodů synchronizace obsahují všechny úkoly v relační databázi tzv. časové razítko v podobě záznamu o datu a času vytvoření a změny úkolu.

Naopak metoda s názvem `AddKvadosWorkItem(KvadosWorkItem inserted, string author)` přijímající přidávaný úkol a jeho autora slouží pro automatickou synchronizaci směrem z TFS do relační databáze. Pokusí se vyhledat přijatý úkol podle jeho `tfs_id` v relační databázi. Pokud ho nalezne a jeho čas poslední změny je starší než čas vkládaného úkolu, tak zavolá metodu pro aktualizaci jeho dat. Pokud se úkol s tímto identifikátorem v relační databázi nevyskytuje, spustí se metoda pro vytvoření zcela nového záznamu.

Funkčnost celého řešení pro automatickou synchronizaci záznamů oběma směry byla odzkoušena, avšak do našeho řešení nakonec zakomponována nebyla. S jejím využitím se počítá během dalšího vývoje.

4.9 Seznámení se s Ext.NET

Po dokončení přípravy v podobě tvorby tříd pro získávání a ukládání dat do relační databáze a TFS se přistoupilo k tvorbě uživatelského rozhraní pro pohodlnou správu přenášených úkolů. Po krátké úvaze bylo zvoleno webové rozhraní založené na technologii ASP.NET pro jednoduchou dostupnost aplikace přes webový prohlížeč z pohledu uživatele a výbornou spolupráci s dříve vytvořeným kódem z pohledu programátora.

Po zvážení, jaké funkce musí klient obsahovat, byla k tvorbě využita mutace ExtJS (Sencha) JavaScript library třetí strany s názvem Ext.NET pro ASP.NET [4]. Ext.NET není výchozí součástí VS. Abychom mohli Ext.NET začít využívat, musí být do projektu napřed přidán. To se provede pomocí NuGet Packages Manager ve VS.

Využité Ext.NET komponenty v rámci tohoto projektu nepodporují náhled pomocí designera ve VS. Vše bylo proto laděno pouze pomocí zdrojového kódu, což se nakonec ukázalo jako prospěšné pro pochopení způsobu práce s touto knihovnou. Díky tomu, že technologie ASP.NET rozděluje aplikační logiku na klientskou a serverovou stranu, bylo snahou obsloužit vše, co se týká uživatelského rozhraní na straně klienta za využití JavaScriptu. Tato snaha přispěla k lepší odezvě celé aplikace.

Jedním z důvodů využití Ext.NET byla i podpora funkce Drag and Drop, která je dnes velmi oblíbená. Bylo rozhodnuto vytvořit dvě okna vedle sebe ve stylu souborových manažerů. Levé okno obsahuje úkoly uložené v relační databázi, kam byly uloženy ze systému VENTUS. Pravé okno obsahuje přehled úkolů uložených v TFS. Primárním úkolem v tuto chvíli bylo proniknout do sémantiky Ext.NET [7] a zprovoznění funkce Drag and Drop pro přesun vybraného úkolu z levého okna do pravého. Po pochopení základních principů Ext.NET a

způsobem práce s potřebnými komponentami opět nic nebránilo postupnému vystavění celé aplikace a případnému obohacení o další funkce.

4.10 Tvorba základního webového GUI a zobrazení dostupných úkolů

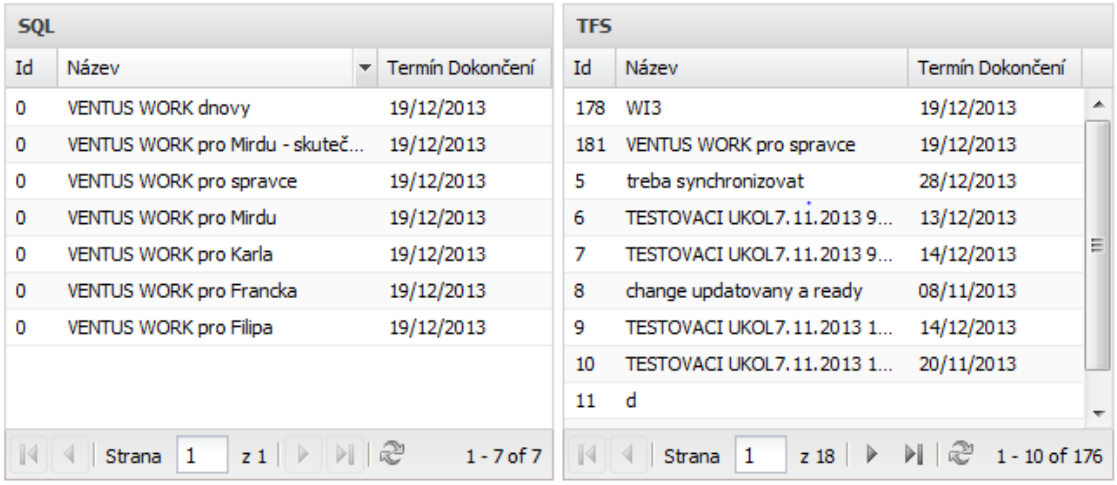
Základním prvkem celé `.aspx` stránky představující naši webovou aplikaci jsou dva elementy typu `GridPanel` tvořící obě okna (Obrázek 4.11), které každé vlastní element `Store` obsahující už konkrétní úkoly. Oba elementy `Store` obsahují element `Model` (Obrázek 4.10) s vnořeným `Fields` v němž jsou jednotlivé `ModelField` s definovanými názvy ukládaných údajů o úkolu. Takto definovaný model se potom aplikuje na elementy `Column` pomocí jejich atributu `DataIndex` reprezentující jednotlivé sloupečky elementu `GridPanel`. Pro zisk úkolů byly v code behind `.aspx` stránky, vytvořeny dvě metody `sql_Load` a `tfs_Load` ve formátu pro obsluhu události. Tento formát byl zvolen, protože jsou využity i pro funkci aktualizace obou seznamů, která bude implementována později. Tyto dvě metody jsou volány při načítání `.aspx` stránky v metodě `Page_Load` k tomu určené. Zde si bylo potřeba pohlídat, aby metody nebyly znovu volány v případě PostBacku. Code behind také obsahuje všechny ostatní .NET metody volané z `.aspx` stránky.

Metoda `sql_Load` získá díky metodě `ReadNewKvadosWorkItems` z třídy `DbWorkItem List` všech úkolů určených pro zobrazení. Položky jednotlivých úkolů přeformátuje do tvaru definovaným v elementu `Store` levého elementu `GridPanel`. Poté je takto upravená kolekce uložena do objektu typu `Store` tohoto panelu. Tímto uživateli zobrazíme úkoly, jenž může do TFS nahrát.

```
<ext:Model runat="server">
    <Fields>
        <ext:ModelField Name="TfsId" />
        <ext:ModelField Name="Name" />
        <ext:ModelField Name="Resolve" />
        <ext:ModelField Name="Description" />
        <ext:ModelField Name="Required" />
        <ext:ModelField Name="State" />
        <ext:ModelField Name="Assigned" />
        <ext:ModelField Name="Remaining" />
        <ext:ModelField Name="Changed" />
        <ext:ModelField Name="TypeName" />
        <ext:ModelField Name="DbId" />
        <ext:ModelField Name="Author" />
    </Fields>
</ext:Model>
```

Obrázek 4.10: Definování ukládaných dat v GridPanelu

Obdobně funguje metoda `tfs_Load`, jenž projde všechny dostupné projekty a získá z nich všechny úkoly pomocí opět pomocí dříve vytvořeným třídám `TfsWorkItemService`, `TfsProjectService` a jejich metodám. Takto získaný a přeformátovaný `List` objektů opět uloží do objektu typu `Store` elementu `GridPanel` umístěného vpravo, jenž je poté zobrazí.



Obrázek 4.11: Základní uživatelské rozhraní s dostupnými úkoly

4.11 Stránkování dostupných úkolů

Uživatelské rozhraní bylo obohaceno o `BottomBar` (Obrázek 4.12) sloužící jako panel pod oběma okny, do kterého bylo přidáno stránkování pomocí `PagingToolbar`. Pro oživení tlačítka aktualizovat, jenž je součástí stránkování stačí elementu `Store`, který součástí stejného elementu `GridPanel` jako `PagingToolbar`, přidat atribut `OnReadData` s názvem metody pro aktualizaci jeho obsahu ve formátu metody pro zpracování události. Z tohoto důvodu byly metody `tfs_Load` a `sql_Load`, načítající úkoly do objektů typu `Store` obou elementů `GridPanel`, takto předem upraveny.

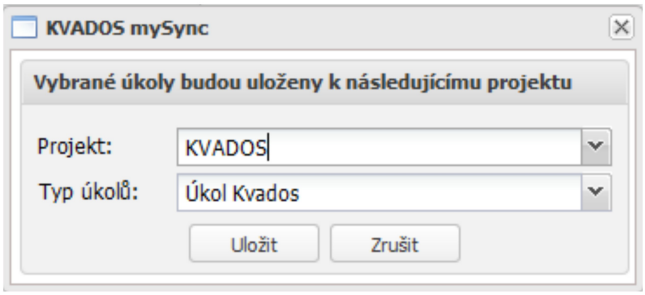
```
<ext:GridPanel
ID="GridPanel1"
runat="server"
MultiSelect="true"
Flex="1"
Title="SQL"
Margins="0 2 0 0">
...
<BottomBar>
<ext:PagingToolbar
runat="server"
DisplayInfo="true"
DisplayMsg="{0} - {1} of {2}"
EmptyMsg="Nic k zobrazení"
RefreshText="Načíst všechny dostupné úkoly"/>
</BottomBar>
</ext:GridPanel>
```

Obrázek 4.12: Funkce stránkování

4.12 Přesun úkolů s možnosti volby projektu

Při přesunu úkolů z relační databáze do TFS není předem stanoveno, ke kterému projektu se úkol vztahuje. Bylo proto vytvořeno dialogové okno `Window`, jehož obsahem je `FormPanel` s několika rozbalovacími seznamy pro určení cílového projektu a tlačítkem pro

potvrzení přenosu. Pro `ComboBox` s možností výběru projektu, k němuž se úkol přiřadí se využívá metoda `LoadProjects` ve formátu zpracování události. Zde se získá pomocí dříve připravené třídy a její metody seznam všech projektů v TFS. Druhý `ComboBox` obsahuje výběr typu přenášených úkolů. Tento formulář (Obrázek 4.13) je nastaven jako skrytý a zobrazí se až po přetažení vybraných úkolů z levého okna do pravého.



Obrázek 4.13: Určení cílového projektu

Pro zrušení možnosti přenášet položky opačným směrem slouží atribut `EnableDrop` nastavený na hodnotu `false` elementu `GridDragDrop` levého elementu `GridPanel`. Po vybrání úkolů a jejich přetažením do pravého elementu `GridPanel` se spustí sled několika úkonů. Připravený `Listener` vyvolá událost `Drop` u pravého elementu `GridPanel` jenž spustí JS kód. Ten si díky JS funkci uloží stranou přenášené úkoly v podobě `var` objektu pod implicitním názvem data získaná interním mechanismem z elementu `Store` levého panelu a zobrazí zmiňované dialogové okno. Stisknutím tlačítka „Uložit“ se pomocí JS Ext.NET funkce `Ext.getCmp('ID ELEMENTU')` získají zvolené možnosti z elementů `ComboBox` a s těmito informacemi jako parametry spustí funkci `sendTasks` (Obrázek 4.14). JS funkce `sendTasks` cyklem projde dříve získané úkoly uložené doposud v objektu data a postupně je všechny předá .NET metodě `SaveItemToTFS`. Po každém odeslaném úkolu informuje o této akci v uživatelském rozhraní tzv. Pop up oknem díky JS funkci `Ext.net.Notification.show`. Všechny metody jako je `SaveItemToTFS` definované v přiřazené třídě a volané z .aspx stránky pomocí Ext.NET a JS musí být označeny atributem `[DirectMethod]`. Zde se z jednotlivých získaných údajů opět sestaví objekt typu `KvadosWorkItem` a pomocí metody `SyncItem` třídy `TfsWorkItemService` uloží do TFS.

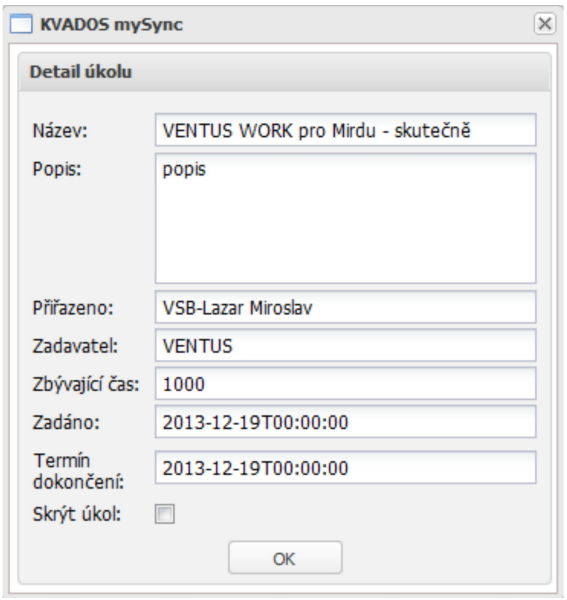
```
function sendTasks(project, taskType)
{
    for (var i = 0; i < data.records.length; i++)
    {
        App.direct.SaveItemToTFS(data.records[i].get('TfsId'),
            data.records[i].get('Name'), data.records[i].get('Resolve'),
            data.records[i].get('Description'), data.records[i].get('Required'),
            data.records[i].get('State'), data.records[i].get('Assigned'),
            data.records[i].get('Remaining'), data.records[i].get('Changed'),
            data.records[i].get('TypeName'), data.records[i].get('DbId'), project,
            taskType, data.records[i].get('Author'));

        Ext.net.Notification.show({
            title: 'Položka odeslána',
            html: 'Task ' + data.records[i].get('Name') + ' úspěšně uložen.'
        });
    }
}
```

Obrázek 4.14: Odeslání vybraných úkolů do TFS

Tímto byla splněna hlavní funkce celé aplikace, takže se přistoupilo k pracím na rozšiřování uživatelského rozhraní o další možnosti.

4.13 Detailní informace o úkolu

V obou elementech `GridPanel` vidí uživatel vždy je jen název úkolu a termín jeho dokončení. Zbylé informace jsou uživateli skryty. Další práce se proto ubíraly tímto směrem. Byl přijat návrh na funkcionalitu, jenž by po dvojkliku na úkol připravený k přenosu v levém `GridPanelu` zobrazila v novém okně detailní informace o tomto úkolu (Obrázek 4.15).



Obrázek 4.15: Detail úkolu

Základem se stal další skrytý `Window` s vnořeným elementem `FormPanel`. Obsahuje elementy `TextField` pro zobrazení jednotlivých údajů, které úkol obsahuje. Jediným ovlivnitelným elementem ve formuláři je `Checkbox` umožňující funkci skrýt úkol v seznamu dostupných úkolů. Okno se zavírá jediným dostupným tlačítkem OK. Funkčnosti vyvolání okna s detaily po dvojkliku na některý z úkolů docílíme přidáním události `ItemDblClick` mezi `Listeners` levého `GridPanelu`. `ItemDblClick` po dvojkliku na vybraný úkol zobrazí doposud skrytý Window a spustí JS funkci `setItemFields` a předá jí celý obsah zvoleného úkolu implicitně zabaleného v objektu jménem `record`. Z tohoto objektu se poté naplní jednotlivé `TextField` díky Ext.NET funkci `Ext.getCmp('IDTextField').setValue(record.get('UkolHodnota'))`.

4.14 Skrytí zvoleného úkolu a opětovné zobrazení

Funkce skrýt úkol v seznamu úkolů byla vyřešena následovně. Kliknutím na tlačítko OK se spustí `Handler`, který spustí JS funkci `setHideItem` a uzavře okno s detailem úkolu. Funkce `setHideItem` zkontroluje zda byl v okně detailu úkolu zaškrtnut `CheckBox`. Zdali tomu tak skutečně je. Provolají se postupně .NET funkce `HideItemDB(string dbId)` definovaná v code behind a `HideItem(string id)`, jenž pro tuto funkčnost rozšířila dříve vytvořenou třídu pro práci s relační databází. Metody si předají z okna detailu úkolu identifikátor zvoleného úkolu a nastaví mu na úrovni relační databáze atribut `hide`. Pro tuto funkčnost musela být tabulka uchovávající všechny úkoly v relační databázi dodatečně rozšířena o tento nový atribut typu BIT určující, které úkoly mají být uživatelům skryty. Nyní již pouze stačilo upravit SQL dotaz v metodě pro získávání dostupných úkolů, aby ignoroval ty, jenž mají příznak `hide` nastaven na '1'.

S obohacením aplikace o výše zmíněnou funkcionalitu bylo třeba zajistit, aby se skryté úkoly daly opět zobrazit. K tomu byl v `.aspx` stránce pod oba elementy `GridPanel` přidán `BottonBar` obsahující `Toolbar` s komponentou `Button`. `Handler` tohoto tlačítka spouští v code behind .NET metodu `ShowItemsDB` s atributem `[DirectMethod]` a znovu načte všechny dostupné úkoly. Pro opětovné zobrazení všech úkolů byla třída pro práci s relační databází dodatečně obohacena o metodu `ShowHiddenItems`, jenž nastavuje pomocí SQL dotazu všem úkolům s nastaveným atributem `hide` na '1' zpět na '0'. Zavoláním této metody z `ShowItemsDB` je splněna i tato funkcionalita.

4.15 Filtrace zobrazených záznamů dle zadaných kritérií

Za stavu, kdy relační databáze a TFS obsahuje velké množství úkolů, se oba seznamy úkolů stávají nepřehlednými. Aby se v seznamu dostupných úkolů lépe orientovalo, byl do uživatelského rozhraní přidán filtr pro selekci úkolů. Lze filtrovat podle: úkoly vybraného uživatele, stavu úkolů a úkoly konkrétního projektu. Volby lze libovolně kombinovat. Filtrace

podle uživatele probíhá jak na straně databáze, tak na straně TFS. Zbylé možnosti filtrují úkoly pouze z TFS.

Nad oba elementy `GridPanel` byl proto vložen `TopBar` obsahující `ToolBar` s trojicí elementů `ComboBox` představujících možnosti filtrování a potvrzovacím tlačítkem `Button`. Pro získání seznamu uživatelů zobrazených v prvním elementu `ComboBox` se rozšířila třída `DbWorkItemService` o novou metodu `GetAllUsers` která vrací `List` objektů všech uživatelů zaznamenaných v relační databázi. Správné namapování hodnot z objektu na `ComboBox` je zajištěno přidáním elementů `Model`, `Fields` a `ModelField` do objektu typu `Store` (Obrázek 4.16) tohoto elementu `ComboBox` a definováním jednotlivých položek přijímaného objektu. `ComboBox` pro výběr podle stavu úkolu je naplněn nejpoužívanějšími typy stavů úkolu pouze staticky pomocí elementu `ListItem`. Ruční nastavení položek bylo zvoleno proto, že se neočekává použití atypických stavů. Poslední `ComboBox` pro výběr projektů je naplňován díky již dříve využité třídě `TfsProjectService` a její metodě `GetTfsProjects`. Metody pro dynamicky naplňované `ComboBoxy` jsou spouštěny při prvotním načítání `.aspx` stránky opět díky metodě `Page_Load`.

```
<Store>
    <ext:Store
        runat="server"
        ID="UserComboStore" >
        <Model>
            <ext:Model runat="server" IDProperty="Id">
                <Fields>
                    <ext:ModelField Name="id" Type="String" ServerMapping="Id" />
                    <ext:ModelField Name="name" Type="String" ServerMapping="FullName" />
                </Fields>
            </ext:Model>
        </Model>
       <Listeners>
            <Load Handler="#{UserCombo}.setValue(#{UserCombo}.store.getAt(0).get('id'));" />
        </Listeners>
    </ext:Store>
</Store>
```

Obrázek 4.16: Definování textu a hodnoty elementu ComboBox

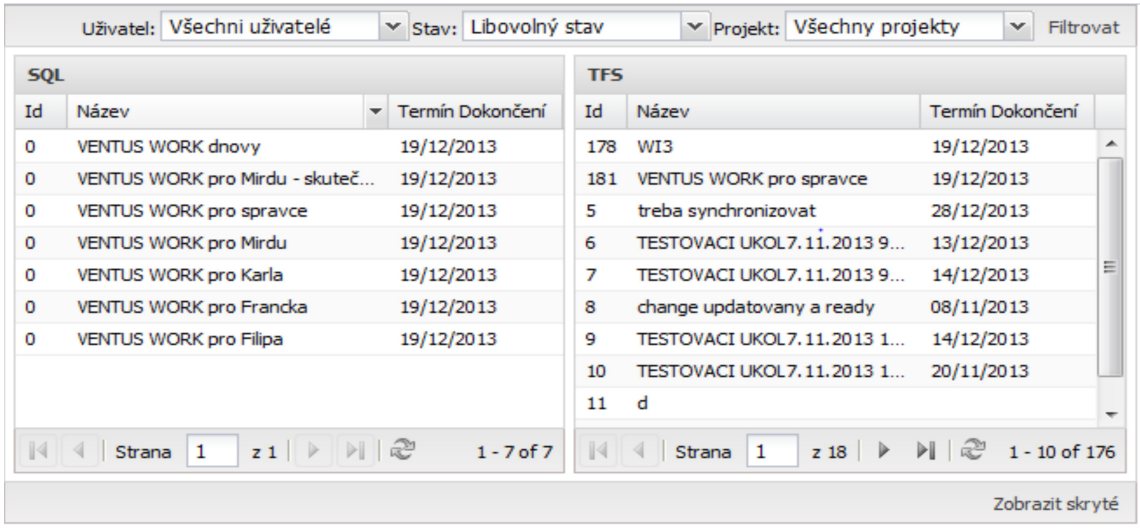
Zisk úkolů podle zadaných preferencí zajišťuje potvrzovací tlačítko a jeho `Handler`. JS zavolá .NET metodu `Filter`, opět povinně označenou atributem `[DirectMethod]`, které předá nastavené hodnoty z elementů `ComboBox` díky Ext.NET JS funkci `getValue('POLOŽKA OBJEKTU')`. Zde se pomocí předem připraveným metodám tříd `TfsWorkItemService` a `DbWorkItemService` získají kolekce úkolů podle získaných kritérií. Tyto kolekce se poté vloží do objektů typu `Store` příslušných elementů `GridPanel`.

4.16 Obohacení filtru o funkci „zobraz vše“

Se zavedením filtru v současném stavu není možno zobrazit v elementech `GridPanel` úkoly všech uživatelů, projektů a s libovolným stavem. Do elementu `ComboBox` filtru pro uživatele, projekty a stavy úkolu byly proto přidány statické elementy `ListItem` s hodnotou „Všichni uživatelé, “„Všechny projekty“, resp. „Libovolný stav“. Na tyto změny musela být přizpůsobena jak metoda `GetKvadosWorkItemsWithState` ze třídy `TfsWorkItemService`, která se stará o zisk seznamu úkolů na straně TFS podle nastavení filtru, tak metoda `getWorkItemsByAssignedTo` třídy `DbWorkItemService` pro zisk úkolu zvoleného uživatele z relační databáze.

Ze všech předpřipravených metod třídy `TfsWorkItemService` pro požadavky filtru přesně vyhovovala `GetKvadosWorkItemsWithState` vracející `List` úkolů z TFS podle vstupních parametrů uživatel, stav, projekt. Dotazování zde vůči TFS probíhá pomocí SQL dotazu s nastavitelnými parametry získanými z hodnot pocházejících z GUI filtru a jeho elementů `ComboBox`. Výběrem možnosti pro zobrazení všech uživatelů, stavů, projektů nabývá za současného stavu SQL dotaz tvaru, jemuž po vyhodnocení neodpovídá žádný úkol z TFS. Původní statický SQL dotaz proto musel být rozložen do více řetězců tvořících část SELECT a jednotlivé WHERE klauzule. Poté je postupně dynamicky skládán. Logiku skládání výsledného SQL dotazu zaručují podmínky, které kontrolují získané parametry metody na výskyt hodnoty značící všechny úkoly, stavy nebo projekty. Když je v jednom z parametrů nalezena hodnota pro označení „zobraz vše“, je podmínka zodpovědná za tento parametr přeskočena a do výsledného dotazu není přidán řetězec s klauzulí WHERE s tímto atributem a parametrem. Podle logiky SQL se poté nebude brát na tento atribut zřetel.

Obdobný problém nastal na straně relační databáze. Metoda `getWorkItemsByAssignedTo` z třídy `DbWorkItemService` získává úkoly připravené k přenosu do TFS podle vybraného řešitele. S přidáním a vybráním volby „Všichni uživatelé“ v elementu `ComboBox` filtru vzniká s SQL dotazem obdobný problém jako popsaný v předešlém odstavci. Díky tomu, že se zde mění hodnota pouze jednoho atributu, není třeba SQL dotaz skládat z jednotlivých řetězců. Řešení je ale jinak obdobné. Do metody byla přidána podmínka, jenž kontroluje parametr vybraného uživatele na výskyt hodnoty značící všechny uživatele. V případě zjištění této hodnoty je parametr `WHERE` klauzule pro atribut uživatel v SQL dotazu nastaven na '%' s využitím operátoru `LIKE`. Operátor s touto hodnotou vyhovuje všem dotazovaným záznamům. Jestliže je zadán konkrétní uživatel, podmínka je přeskočena a za parametr `WHERE` klauzule je vložen tento uživatel.



Obrázek 4.17: Finální vzhled uživatelského rozhraní

S vyřešením všech problémů popsaných výše byly na uživatelském rozhraní (Obrázek 4.17) dokončeny všechny vyžadované práce.

4.17 Možnosti ověření identity v ASP.NET s využitím Active Directory

Po otestování všech možných stavů, které mohou po implementací filtru nastat, bylo přikročeno k realizaci jedné z posledních důležitých funkcí aplikace. Za současného stavu mohl kdokoliv přiřadit libovolný úkol k libovolnému projektu. Myšlenka byla vytvořit přihlašování uživatelů vůči Active Directory. Tento způsob přihlášení je vhodný z důvodu, že TFS řeší správu uživatelů také pomocí AD. Přihlášený uživatel by poté mohl pouze prohlížet a přidávat nové úkoly pouze k projektům, kterých je členem.

Ověření identity vůči AD pro potřeby aplikace je možno v ASP.NET řešit velice jednoduše v několika krocích. [6] Do souboru `web.config` přidáme nový `connectionString` ve formátu:

```
<connectionStrings> <add name="NAZEV PRIPOJENI"
connectionString="LDAP://company.com.au/DC=company,DC=com,DC=au
"/></connectionStrings>
```

Pod něj do sekce `<system.web>` umístíme Membership provider ve tvaru:

```
<membership defaultProvider="MyADMembershipProvider">
<providers>

<add name="MyADMembershipProvider"
type="System.Web.Security.ActiveDirectoryMembershipProvider,
System.Web, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a" connectionStringName="NAZEV
```

```
PRIPOJENI" attributeMapUsername="sAMAccountName"/>
</providers></membership>
```

Pro funkci přihlášení pomocí webového formuláře je potřeba dále do sekce `<system.web>` přidat autentizační a autorizační parametry:

```
<authentication mode="Forms">
<forms name=".ADAuthCookie" timeout="43200"/>
</authentication> <authorization> <deny users="?"/>
<allow users="*"/> </authorization>
```

4.18 Tvorba přihlašovacího GUI

Tímto je konfigurace dokončena a ASP.NET očekává přihlašovací stránku s názvem `Login.aspx`. Tato stránka byla vytvořena a nastavena jako „Start Page“. Zde se využilo základních elementů ASP:NET `Label` a `TextBox` pro zadání jména a hesla. Ověření vůči AD se provede implicitně přes element `Button` po přidání atributu `CommandName` s hodnotou `Login`. Když se ověření podaří, je uživatel ve výchozím nastavení přesměrován na stránku `default.aspx`. K přihlašovacímu tlačítku byla dále přidána událost `OnClick` a v code behind vytvořena metoda která jí zpracovává. Zde se kontroluje, zda jsou zadávané údaje do přihlašovacího formuláře korektní a při nalezení chyby je uživatel zpětně informován. Také se zde po úspěšném ověření uživatele a před přesměrováním na stránku `default.aspx` s GUI do `Session` uloží zadané uživatelské jméno pro identifikaci přihlášeného uživatele. Do metody `Page_Load` v code behind stránky `default.aspx` byl přidán příkaz pro opětovné získání přeposlaného uživatelského jména pomocí `Session`.

4.19 Modifikace tříd pro ověření přístupu k projektu

Nyní, když známe identitu uživatele, je potřeba upravit třídy pro získávání dat z TFS, aby získávaly jen úkoly a projekty přihlášeného uživatele. Entitní třída `KvadosProject` byla rozšířena o vlastnost `Members`, což je `List` řetězců uživatelských jmen z AD zapojených do projektu. Třída `TfsProjectService` byla obohacena o privátní metodu vracející `List` uživatelských jmen zadaného projektu, který se nyní vkládá v `GetTfsProjects` do vlastnosti `Members` aktuálně načítaného projektu. Tento `List` je poté porovnáván s nově přidaným parametrem metody zastupující jméno přihlášeného uživatele. Pokud se jméno uživatele v parametru vyskytuje mezi členy projektu, tak se tento projekt přidá mezi projekty, které celá metoda vrací.

4.20 Modifikace filtru pro spolupráci s ověřením přístupu

Nyní po přihlášení budou uživateli zobrazeny pouze ty projekty, na kterých pracuje a jejich úkoly. Pokud se však za současného stavu využije funkce filtru pro zobrazení úkolů všech projektů. Budou zobrazeny všechny bez ohledu zda je přihlášený uživatel jejich členem nebo není. Již několikrát upravovaná metoda `GetKvadosWorkItemsWithState` z třídy `TfsWorkItemService` byla z tohoto důvodu rozšířena o nový parametr přihlášeného uživatele. V části kde se skládá SQL dotaz již nestačí v případě požadavku pro zobrazení úkolů všech projektů tuto `WHERE` klauzuli pouze přeskočit. Nově byla přidána alternativní větev podmínky pro test hodnoty pro zobrazení úkolů všech projektů. Zde se získá pomocí `GetTfsProjects` z třídy `TfsProjectService List` všech dostupným projektů pro přihlášeného uživatele. Z tohoto seznamu se následně poskládá `WHERE` klauzule pro výsledný SQL dotaz.

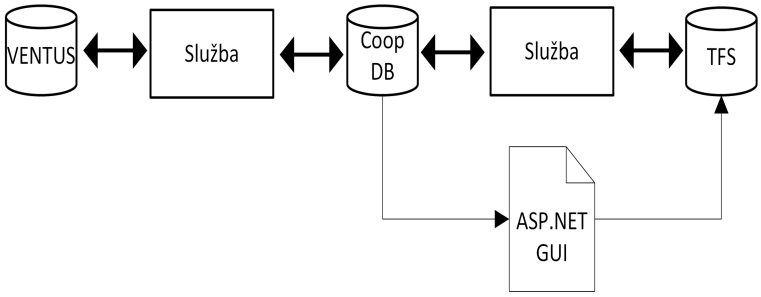
4.21 Sloučení vyvíjených projektů do společného řešení

Dokončením prací na funkci přihlašovaní uživatelů byly splněny všechny funkční požadavky na aplikaci v rámci bakalářské praxe. Mnoho z připravených metod pro zisk úkolů vyhovujícím nejrůznějším požadavků z relační databáze nebo z TFS nebylo nasazeno. Funkce automatické synchronizace do TFS a následná zpětná aktualizace záznamů relační databáze v pravidelných intervalech upravených úkolů byla otestována, avšak do aplikace zatím nezahrnuta. Všechny tyto nevyužité metody zůstaly zachovány pro pozdější využití při plánovaném dalším rozšiřování celé aplikace.

Za těchto podmínek se zahájilo sloučení doposud odděleně tvořených částí celé aplikace do společného řešení. Entity a třídy pro získávání dat byly uloženy jako samostatná knihovna tříd, na které odkazuje projekt webové aplikace. Do tohoto řešené přidal svou práci zodpovědnou za získávání dat ze systému VENTUS také spolužák a spolupracovník Karel Antošík.

4.22 Plán nasazení

Funkčnost finálního řešení bude následovná (Obrázek 4.18). Služba běžící na pozadí zodpovědná za společnou a VENTUS databázi bude v pravidelných intervalech kontrolovat změněné položky v obou databázích od posledního data kontroly. Tyto položky poté mezi databázemi synchronizovat. Tuto část vytvářel již zmíněný spolužák Karel Antošík.



Obrázek 4.18: Schéma finálního řešení

Pro přesun úkolů ze společné databáze do TFS slouží primárně dříve popsaný ASP.NET klient. Po určení cílových projektů pro zvolené úkoly a jejich přesunu do TFS však bude do budoucna potřeba vložené úkoly v případě jejich dodatečné úpravy udržovat mezi databázemi aktuální. K tomu bude sloužit opět služba s podobným principem, který používá služba na straně pro práci se systémem VENTUS. Pro tuto službu je vše připraveno a otestováno, avšak její funkčnost není cílem práce této odborné praxe.

5. Znalosti uplatněné a získané během praxe

Programovacímu jazyku C#, ve kterém celý vývoj probíhal, se aktivně věnuji již od střední školy. Na vysoké škole jsem si tyto znalosti prohloubil díky předmětu *Programovací Jazyky II*. Proto zdánlivě nic nebránilo zahájení přípravných prací na první části vyvíjené aplikace. Avšak TFS pro mě, jakožto doposud sólového programátora, byl zcela neznámý systém. Při jeho provázanosti s VS jsem byl překvapen, že jsem se s ním za celou dosavadní dobu studia nesetkal. V tomto mi praxe poskytla nenahraditelný praktický pohled do týmového vývoje softwaru a samotného fungování TFS.

Technologie a postupy spojené s dotazovacím jazykem SQL a tvorbou relačních databázových systémů na platformě SQL Server mi již byly známy z vysokoškolského studia díky předmětům *Úvod do Databázových Systémů* a *Databázové a Informační Systémy*. Tyto předměty svým zaměřením a vyšší náročností poskytly solidní základy. Avšak až tato stáž ukázala, zda nabyté vědomosti jsou uplatnitelné v praxi.

Technologii ASP.NET využité v posledních měsících praxe jsem se již nějaký čas věnoval výhradně samostudiem. Během mého bakalářského studia byl ASP.NET v několika předmětech zmíněn, bohužel jen okrajově. To mi přijde velice nešťastné, hlavně kvůli jeho budoucímu potenciálu v podobě dnešního trendu přesunu aplikací na web. ASP.NET stírá rozdíly mezi desktopovou aplikací a webem. Využití dodatečně instalovaných modulů bylo pro mě novinkou. Po integraci Ext.NET do nového projektu ASP.NET díky NuGet Package Manager přišla vhod velmi bohatá dokumentace s příklady, kterou Ext.NET nabízí na svých webových stránkách.

JavaScriptu jsem se samostatně nikdy nevěnoval z důvodu zaměření pozornosti na desktopový vývoj. Avšak pro své nenahraditelné místo ve webových technologiích ho nešlo dále přehlížet. Díky skutečnosti, že jsem v té době dokončoval předmět *Vývoj Internetových Aplikací*, kde byl JavaScript výrazně zastoupen po celou dobu studia, nebylo hledání materiálů a pochopení výraznějším problémem a celkově mi tento předmět a tato praxe vzájemně pomohly pro snadnější pochopení JavaScriptu i jeho využití v Ext.NET.

6. Závěr

Absolvování odborné praxe bylo pro mě výbornou životní zkušeností ve všech směrech. Ať už šlo o technické stránky a postupy při implementaci pomocí programovacích jazyků nebo mezilidskou komunikaci ve firemním prostředí. Během praxe se také potvrdila moje myšlenka, s kterou jsem na tuto praxi nastupoval. Je veliký rozdíl mezi pasivním věděním a schopností využit tyto nabité znalosti ze studia při řešení konkrétních úkolů. Což v oboru IT platí dvojnásob. Při nepřeberném množství odvětví IT průmyslu mi tato praxe současně pomohla s úvahami, jakým směrem se dále vydat, v čem se nadále zdokonalovat a jakým technologiím se věnovat.

Vyvíjený nástroj byl vystavěn zcela od základu a postupně obohacován o další aktuálně požadované funkcionality bez hlubší počáteční analýzy. Zde se využilo agilního vývoje a metodik s ním spojených. Samotné nasazení nástroje nebylo během praxe provedeno. Výsledný produkt je v plánu dále rozšiřovat a posléze využít pro zefektivnění interních firemních procesů. Pokud k tomu dojde, bude to důkaz, že absolvovaná praxe byla přínosem nejenom pro mě. Také se ukázalo, že žádný problém, ať už vypadá sebesložitěji, není neřešitelný, pokud se rozdělí do menších podúkolů.

7. Použitá literatura

[1] KVADOS a.s.: O Společnosti. *KVADOS software solutions* [online]. 2014. [cit. 2014-04-10]. Dostupné z: http://www.kvados.cz/Content/O-spolecnosti

[2] VENTUS: Software. *KVADOS software solutions* [online]. 2014. [cit. 2014-04-10]. Dostupné z: http://www.kvados.cz/Content/VENTUS-Software

[3] Team Foundation Server 2013: Platform. *Microsoft* [online]. 2014. [cit. 2014-04-10]. Dostupné z: http://msdn.microsoft.com/en-us/vstudio/ff637362.aspx

[4] Ext.NET: Library. [online]. 2014. [cit. 2014-04-10]. Dostupné z: http://www.ext.net

[5] Komunikace mezi Visual Studio a TFS: .NET assemblies. *Developer Network* [online]. 2014. [cit. 2014-04-10]. Dostupné z: http://msdn.microsoft.com/en-us/library/bb130146.aspx

[6] Using Active Directory to authenticate users to your ASP.NET Web Site : topic. [online]. 2014. vyd. [cit. 2014-04-10]. Dostupné z: http://evonet.com.au/using-active-directory-to-authenticate-users-to-your-asp-net-web-site/

[7] SHAH, Anup. *Ext.NET Web Application Development*. Birmingham: Packt Publishing, 2012. ISBN 978-1-84969-324-0.